Executable Graphics for PBNM

R. Lopes[1], N. Raimundo[1] M. Varanda[1], J. Oliveira[2], and V. Roque[3]

[1] Polytechnic Institute of Bragança, 5301-854 Bragança, Portugal
rlopes,neves,mjoao@ipb.pt
[2] University of Aveiro 3810-193 Aveiro, Portugal
jlo@det.ua.pt
[3] Polytechnic Institute of Guarda 6301-559 Guarda, Portugal
vitor.roque@ipg.pt

Abstract. The specification of a policy is performed in a policy language, usually following a textual representation. However, humans process images faster than text and they are prepared to process information presented in two or more dimensions: sometimes it is easier to explain things using figures and their graphical relations than writing textual representations.
This paper describes a visual language, in the form of graphics that are executed in a networking environment, to define a network management policy. This approach allows to map visual tokens and corresponding arrangements into other languages to which a mapping is defined.

1 Introduction

Network management has been a constant worry among organisations and network operators in the last decades. We have seen several approaches being developed and proposed, from distributed systems solutions (such as CORBA – JIDM, for example – or Mobile Agent based solutions) to specific solutions, such as the SNMP or CMIP. Among them, the SNMP model has, probably, been the most well known and widely used. However, none of them has fully satisfied the community which is still searching for an appropriate model or paradigm that can be used efficiently in network management scenarios.

Policy-Based Network Management (PBNM) has become a promising solution for managing enterprise-wide networks and distributed systems. It is targeted to systems that are dynamic in nature and where stopping and recoding is undesirable – changing the policy rules allows the system to modify its behaviour [1].

The PBNM paradigm has proved, at least in theory, that is a good solution for network management. The ideas involved in this paradigm helps greatly network managers in the complicated tasks of the network administration. In fact, when all the theoretical PBNM concepts are applied to practical and effective management applications, the network administration task will be much easier in terms of time, money and difficulty.

Central to the PBNM is the concept of *policy*, usually considered the link between high-level business specification of desired services and low-level device

configurations that provide those services [2]. This definition implies some form of communication and, consequently, a means of describing the concepts associated with the business-level goals – a *policy description language*.

Currently, there are several languages that can be used in policy definition. Some authors have already discussed some of them and have also presented new approaches [3, 4]. However, most of the approaches rely on formalisms that are hard to remember and, sometimes, hard to use, forcing the user to learn new terms and even new constructions. Much of this effort can be reduced by using visual languages, where the user combines pictorial elements to build a flowchart like arrangement or other similar structure [5].

In this paper, we propose the construction of a graphical editor to describe policies. The idea is to create a visual language to specify policies and than use an editor for generating the data structure that can be recognised by the network as a generic representation of a policy. In this context, we are using the PCIM formalism to store and to install the policy in the network.

2 Visual Languages for PBNM

The purpose of a policy description language is to translate from a business specification, such as those found in a Service Level Agreement (SLA), to a common vendor and device-independent intermediate form.

Although there are several approaches and formalisms for specifying policies, there is a common understanding on the concepts involved [1, 6, 3]. This implicit "understanding" allows the specification of a common representation of policies. In general, we can classify policies in two broad classes: *configuration* policies and *management* policies. Configuration policies are used to define initial, or otherwise *condition independent*, rules, used in the configuration of resources and in the definition of state independent policies. As an example, consider the following sentence:

"file 'X' can be accessed by users from group 'students' " (1)

Management policies can be used to define adaptable management actions, usually based on *event-triggered, condition-action* rules. For example:

"notify the admin when the error rate of outgoing packets is increasing" (2)

More focused in the latter approach, the IETF together with the DMTF, has done a remarkable work with the Policy Core Information Model (PCIM) [7] and corresponding extensions – PCIMe [8]. The Policy Framework WG defines *policy* as an aggregation of policy rules [2]. Each policy rule is made up of a set of conditions and a corresponding set of actions. Although this type of policy rule does not explicitly specify an event to trigger the execution of the actions, it assumes an implicit event, such as a process being lauched or a particular traffic flow. In this case, the rules will include an event part [9]:

[(*policyEvent*) causes] (*policyAction*) if (*policyCondition*) (3)

Grouping is also implicit, as it represents the aggregation of related objects. Finally, each network component may represent or act in a role or in a set of roles. The role is a label which indicates a function of an interface or device. In other words, a policy is a set of rules, which can belong to one ou more groups, and which can be applied to devices acting under a specific role.

2.1 Using PBNM Languages

At the highest level, policies may resemble a human language, such as: "user xpto may transfer files". At system level the same policy may assume a different form, still device and technology independent. Usually, policies are specified in a format which is relatively easy to convert to network configuration commands. This paper does not intend to be a survey of the existing approaches, however it is important to provide some insight into how some of the languages are and how do they look like.

Several languages have been developed, resulting from the effort of the academia as well as private enterprises, such as IBM or Sun. Probably, one of the most popular is the Ponder Policy Specification Language [10]. Ponder is a declarative, object-oriented language that can be used to specify both configuration and management policies. It supports obligation policies that are event triggered condition-action rules for policy based management of networks and distributed systems. Key concepts of the language include domains, to group the object to which policies apply, roles to group policies relating to a position in an organisation, relationships to define interactions between roles and management structures to define a configuration of roles and relationships pertaining to an organisational unit such as a department.

The following example shows how Ponder can be used in policy description:

```
inst auth+ filter {
  subject /Agroup + /Bgroup;
  target UAStaff - DETUAgroup;
  action VideoConf(BW, Priority)
    {in BW=2; in Priority=3;} // default filter
    if(time.after("1900")) {in BW=3; in Priority=1;}
}
```

The above policy says that the members of the predefined groups `Agroup` and `Bgroup` may use the video-conference service to the group `UAStaff` excluding `DETUAgroup`. If the service is used after 7 PM, the bandwidth is set to 3Mb/s with priority 1. In other circumstances, the bandwidth is 2Mb/s and the priority is 3.

XACML [11] is an XML specification for expressing policies mainly dedicated to access control and is being defined by the Organisation for the Advancement of Structured Information Standards (OASIS). The language supports roles, which are the same as groups, and are defined as collections of attributes relevant to a principal. It includes conditional authorization policies, as well as policies

with external post-conditions to specify actions that must be executed prior to permitting an access.

The following example describes a scenario where a set of video streaming servers offers tutorials to registered and unregistered users. Registered users have permission to access any server offering a service without time restrictions. Unregistered users can have access to the video-streaming service only from the internal network and not in business-time [12].

```
<service serviceId="TutorialVideoStreaming">
 <description>tutorial video-stream</description>
 <sap>
  <inetaddress> 192.168.200.10 </inetaddress>
  <inetaddress> 192.168.5.3 </inetaddress>
  <protocol>tcp</protocol>
  <port>8976</port>
 </sap>
 <serviceLevel serviceId="Gold">
  <ResourceRsvp AttributeId="qosG7" RsvpClass="G7">
   <TspecBucketRate_r>9250</TspecBucketRate_r>
   <TspecBucketSize_b>680</TspecBucketSize_b>
   <TspecPeakRate_p>13875</TspecPeakRate_p>
   <TspecMinPoliceUnit_m>340</TspecMinPoliceUnit_m>
   <TspecMaxPacketSize_M>340</TspecMaxPacketSize_M>
   <RsvpService>Guaranteed</RsvpService>
   <RsvpStyle>FF</RsvpStyle>
  </ResourceRsvp>
 </serviceLevel>
 <serviceLevel serviceId="Silver"> . . .  </serviceLevel>
 <serviceLevel serviceId="Bronze"> . . .  </serviceLevel>
</service>
```

The Routing Policy System WG of the IETF has defined the RPSL (Routing Policy Specification Language) [13]. It was one of the first languages for specifying routing policies and aims at generating router configuration from the policy specification [14]. A possible example is:

```
aut-num:        AS2
as-name:        CAT-NET
descry:         Teste
import:         from AS1 accept ANY
import:         from AS3 accept <^AS3+S>
export:         to AS3 announce ANY
export:         to AS1 announce AS2 AS3
admin-c:        AO36-RIPE
tech-c:         CO19-RIPE
mnt-by:         OPS4-RIPE
changed:        estig@ipb.pt
source:         RIPE
```

The IETF did not define a specific language to express network policies but rather a generic object-oriented information model for representing policy information. An advantage of the information modelling approach followed by the IETF is that the model can be easily mapped to structured specifications such as XML, which can then be used for policy analysis as well as distribution of policies across networks. The mapping of CIM to XML is already undertaken within the DMTF [15]. The IETF has defined a mapping of the PCIM to a form that can be implemented in a directory that uses LDAP as its access protocol [16]. Considering the following example, we will try to specify the attributes of all the necessary PCIM classes.

```
Group students: Role=[studentPrinters] {
  if (studentPrinterQuota < 0) {
    deny printing job;
  }
}
```

The first thing to do is to create an instance of `CIM_PolicyRule` (Table 1) to define the base of the policy. Then, the condition is created by defining an

CIM_PolicyRule	
Caption	"Policy"
CommonName	"printer quota"
ConditionListType	'DNF'
CreationClassName	"CIM_PolicyRule"
Description	"Controls printing jobs"
ElementName	"printer quota"
Enabled	'Enabled'
ExecutionStrategy	'Do Until Failure'
PolicyDecisionStrategy	'First Matching'
PolicyRuleName	"printer quota"
RuleUsage	"Test the printer quota"
SequencedActions	'Don't Care'
PolicyKeywords	'USAGE'

Table 1. PolicyRule

CIM_VendorPolicyCondition	
Caption	"Condition"
CommonName	"printer quota"
ConstraintEnconding	"UTF-8"
CreationClassName	"VendorPolicyCondition"
Description	"Test students print jobs"
ElementName	"Test print quota"
PolicyConditionName	"Test print quota"
PolRuleCrtnClassName	"CIM_PolicyRule"
PolicyRuleName	"printer quota"
Constraint	"studentPrinterQuota<0"
PolicyKeywords	'USAGE'

Table 2. VendorPolicyCondition

CIM_PolicyConditionInPolicyRule	
ConditionNegated	'FALSE'
GroupNumber	1
GroupComponent	"printer quota"
PartComponent	"Test print quota"

Table 3. PolicyConditionInPolicyRule

instance of `CIM_VendorPolicyCondition` (Table 2).

After creating the condition, it must be associated to the policy with an instance of `CIM_PolicyConditionInPolicyRule` (Table 3). Now the action is specified by creating an instance of `CIM_ActionPolicyCondition` (Table 4) and it is associated to the policy through an instance of `CIM_PolicyActionInPolicyRule` (Table 5).

The resulting policy must be associated to a group, so we need to create the group (Table 6) and create the association object (Table 7). Finally, the policy must be set to a role (Tables 8 and 9).

CIM_VendorPolicyAction	
ActionEncoding	“UTF-8”
Caption	“Action”
CommonName	“printer quota”
CreationClassName	“CIM_VendorPolicyAction”
Description	“Deny printing job”
DoActionLogging	“Denying printing job”
ElementName	“Deny printing”
PolicyActionName	“Deny printing”
PolRuleCrtnClassName	“CIM_PolicyRule”
PolicyRuleName	“printer quota”
ActionData	“deny printing job”
PolicyKeywords	‘USAGE’

Table 4. VendorPolicyAction

CIM_PolicyActionInPolicyRule	
ActionOrder	1
GroupComponent	“printer quota”
PartComponent	“Deny printing”

Table 5. PolicyActionInPolicyRule

CIM_PolicyGroup	
Caption	“Policy Group”
CommonName	“printer quota”
CreationClassName	“CIM_PolicyGroup”
Description	“policy group”
ElementName	“Students”
Enabled	‘Enabled’
PolicyDecisionStrategy	‘First Matching’
PolicyGroupName	“Students”
PolicyKeywords	‘USAGE’

Table 6. PolicyGroup

CIM_PolicySetComponent	
Priority	4
GroupComponent	“printer quota”
PartComponent	“Students”

Table 7. PolicySetComponent

CIM_PolicyRoleCollection	
Caption	Role
Description	“printers”
ElementName	“Role1”
InstanceID	“IPB:role1”
PolicyRole	“studentPrinters”

Table 8. PolicyRoleCollection

CIM_PolicySetInRoleCollection	
Collection	“IPB:role1”
Member	“printer quota”

Table 9. PolicySetInRoleCollection

A clear message resulting from the above paragraphs is that the textual description of policies is a boring and error prone task. With these semantics, the primary goal of PBNM may be compromised since the idea is to simplify tasks through the use of high level, business oriented, declarations. The assistance of a graphical user interface, that can help in filling the various attributes, may simplify the job and may allow a faster definition of policies. We can establish a parallel between the Visual Basic programming environment, where the development of applications is faster and may be achieved by less skilled programmers, and a C++ programming environment, where the development of applications typically takes more time but results in a better structured code.

2.2 Using Visual Languages

A visual language is characterized by the use of graphical notation – a graphical vocabulary and sentences constructed by a spacial combination of symbols (two dimensions). There are several reasons to use visual languages. The basic idea is that humans process images faster than text and that they are prepared to process information presented in two or more dimensions.

The human reasoning is image-oriented: sometimes it is easier to explain things using figures and their graphical relations than writing textual representations. A text has a sequential structure and its visual aspect is always the

same. When we read a text we must understand each character to read a word and we must understand all the words to read and understand sentences.

In visual languages each figure can give different information depending on its size, on its colour, on its form and so on. A visual sentence can be more attractive and easier to understand. So, it is common to use drawings to explain things. Many pedagogical tools use visualisations and animations as a simple way to achieve their purposes. However, visual languages are not as simple to specify, process or traduce by automatic mechanisms than textual languages.

There are some visual language compiler generators and other interesting tools (some of them are very specific) that can be used to process visual languages in order to generate other representations.

The idea of using a visual language to specify policies allows the user to work with a graphical editor to build a structure that can be interpreted as a policy. Just like with textual languages, where compiler-compilers, such as LEX and Yacc, build the code to deal with some programmer defined language, graphical languages can also be achieved by compiling a grammar to build an editor automatically.

3 Executable Graphics for PBNM

In the previous sections we described some approaches to specify policies through the use of textual languages. This section presents a visual language, composed of graphical elements which are executed into a textual description. Because we like to interpret the visual richness of objects as executable charts we use the term *executable graphics* [17].

The main idea is to describe the policy language using a grammar, to be compiled into a graphical editor, and the appearance of the tokens using images: :The advantages of formal specification of programming language semantics are well known: first, the meaning of a program is precisely and unambiguously defined; second, it offers a unique possibility for automatic generation of compilers or interpreters.

```
screen     : policy resources
resources  : resources resource
           | resource
resource   : NAME roles
policy     : rule roles
roles      : roles role
           | role
role       : STRING
rule       : IF condition THEN actions
           | IF condition THEN actions WHEN events
condition  : dnf | cnf
dnf        : exp1
           | dnf OR exp1
cnf        : exp2
           | cnf AND exp2
```

```
exp1       : cterm AND cterm
exp2       : cterm OR cterm
cterm      : STRING
           | NOT STRING
actions    : actions action
           | action
action     : STRING
events     : events event
           | event
event      : STRING
```

We have chosen a spatial paradigm for the policy definition instead of a chart like approach because we think it is easier to fill spaces than connecting lines. The user will need to drag and drop elements to specific spaces in the user interface, thus making the definition of policies easier. We assume that the non-terminal symbols (`role`, `cterm`, `action` and `event`) are represented as blocks in the user interface in a corresponding pallet. This tablet is the initial drag point and the source for the conditions, actions, events and roles.

A condition is a combination of condition terms (`cterm`) in CNF or DNF form. Considering that we want to build a conjunction of two conditions ($Condition1 \wedge Condition2$) it is necessary to drag and drop both of them into the same screen space. The user interface will show the resulting condition inside the same block. If the condition is negated, it is shown as a patterned box with diagonal lines. If we further want to have the result of a conjunction of three more conditions in DNF form, we need to build a disjunction of the resulting blocks (Figure 1). The colours represent the type of operation. A light colour (light gray, in this case) represents the AND operation and the darker colour represents the OR operation. Using CNF instead of DNF would reverse the colours, as shown in Figure 2.

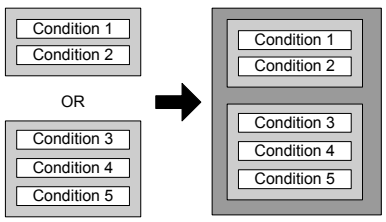

Fig. 1. Building a condition.

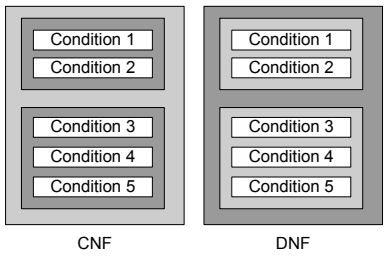

Fig. 2. CNF vs. DNF.

Actions can be sequenced or not. The former implies that the policy, when the condition is true, executes the actions in the order defined by the user. In this situation, as the user drops the actions in the specific place, the actions are connected with arrows.

We considered that the policy can be triggered by one or more events, so we defined a dark area to allow the user to drop a disjunction of events ($Event1 \vee Event2 \vee Event3$). The resulting interface groups roles, conditions, actions and events in a logical distribution and with well defined spatial distribution (Figure 3).

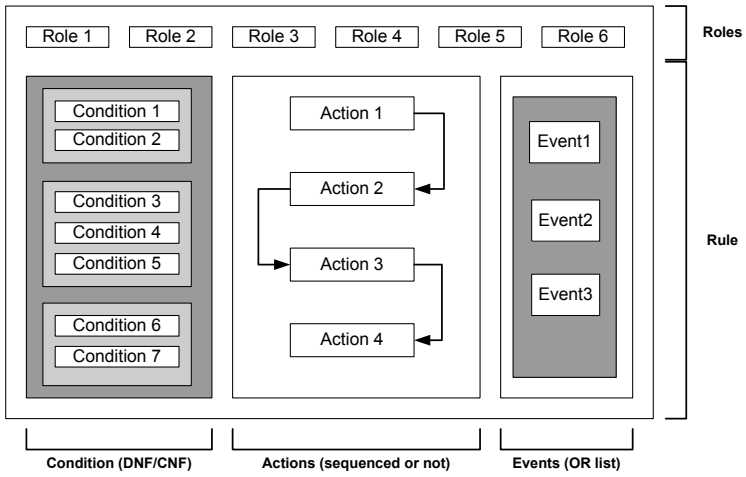

Fig. 3. Visual specification of a policy.

4 Conclusions

The way users communicate with computational entities is usually based on some form of language. Textual languages, although easier to process by automatic processes, are seldom as user friendly as its visual equivalents. A visual sentence can be more attractive and easier to understand. In fact, humans recall faster, and understand better, a subject through some form of visual representation – it is common to use drawings to explain things.

We have defined a visual language for the definition of network management policies. This language works by dragging blocks, representing the actions, conditions and events, and dropping them in specific spaces for building a policy rule. This approach helps reducing the number of attributes to set as well as providing the user with visual aid in remembering the policy components. A well known analogy is the development of applications in a visual language, such as Visual Basic. Although less structured, it helps reducing the time-to-market and the rapid prototyping of applications when compared to textual languages, such as C++.

We hope that this approach helps reducing the difficulty in specifying and remembering policies which, some times, are written in a form difficult to understand and to remember.

References

1. Sloman, M.: Policy driven management for distributed systems. Journal of Network and Systems Management **2** (1994)
2. Westerinen, A., Schnizlein, J., Strassner, J., Scherling, M., Quinn, B., Herzog, S., Huynh, A., Carlson, M., Perry, J., Waldbusser, S.: Terminology for Policy-Based Management. RFC 3198, IETF (2001)
3. Stone, G., Lundy, B., Xie, G.: Network policy languages: A survey and a new approach. IEEE Network **15** (2001) 10–21
4. Damianou, N.: A Policy Framework for Management of Distributed Systems. PhD thesis, Imperial College (2002)
5. Shneiderman, B.: Direct manipulation. a step beyond programming languages. IEEE Transactions on Computers **16** (1983) 57–69
6. Wies, R.: Policies in network and system management – formal definition and architecture. Journal of Network and Systems Management **2** (1994)
7. Moore, B., Ellesson, E., Strassner, J., Westerinen, A.: Policy Core Information Model – Version 1 Specification. RFC 3060, IETF (2001)
8. Moore, B., Ed.: Policy Core Information Model (PCIM) Extensions. RFC 3460, IETF (2003)
9. Chomicki, J., Lobo, J., Naqvi, S.: Conflict resolution using logic programming. IEEE Transactions on Knowledge and Data Engineering **15** (2003)
10. Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The ponder policy specification language. In: Policy 2001: Workshop on Policies for Distributed Systems and Networks, Bristol, UK, Springer-Verlag (2001)
11. OASIS: extensible access control markup language (xacml) version 2.0. Technical report, OASIS (2005)
12. Toktar, E., Jamhour, E., Maziero, C.: Rsvp policy control using xacml. In: IEEE 5th International Workshop on Policies for Distributed Systems and Networks (POLICY 2004), New York (2004)
13. Alaettinoglu, C., Villamizar, C., Gerich, E., Kessens, D., Meyer, D., Bates, T., Karrenberg, D., Terpstra, M.: Routing Policy Specification Language (RPSL). RFC 2622, IETF (1999)
14. Meyer, D., Schmitz, J., Orange, C., Prior, M., Alaettinoglu, C.: Using RPSL in Practice. RFC 2650, IETF (1999)
15. DMTF: Representation of cim in xml (xml mapping specification), v2.0.0. Technical report, DMTF (1999)
16. Strassner, J., Moore, B., Moats, R., Ellesson, E.: Policy Core Lightweight Directory Access Protocol (LDAP) Schema. RFC 3703, IETF (2004)
17. Lakin, F.: Spatial parsing for visual languages. In Chang, S.K., Ichikawa, T., Ligomenides, P., eds.: Visual Languages. Plenum Press, New York (1986) 35–85